CHAPTER

9

HARDWARE REDUNDANCY AND FAULT TOLERANCE

After reading this chapter and completing the exercises, you will be able to:

- Prepare for emergencies that can cause system downtime
- Manage the power supply to your Linux computer
- Check the integrity of your Linux file systems
- Understand how redundant disk systems can protect data

In the previous chapter you learned how to set up and manage user accounts and multiple file systems in Linux, which included creating new user accounts and new file systems. You also learned basic information about how to view and manage processes in Linux.

In this chapter you learn how to protect the data on your Linux system from downtime. This involves planning for potential trouble before it occurs and understanding how to manage key aspects of a Linux computer to avoid downtime. As you will learn in this chapter, the power supply and hard disks are key potential causes of downtime.

PREPARING FOR EMERGENCIES

As you probably know by now, computer systems are not infallible. For all the dependence society places on computers, they continue to break down, leading to crises for customers, managers, and system administrators. These crises loom largest for system administrators, the people everyone else looks to for immediate guidance and resolution when an organization's computer systems malfunction. These are the times when a system administrator who has prepared for emergencies really shines. Although handling crises is part of the system administrator's job description, those who excel in this area are on the path to promotions or raises when the proverbial sky falls. Conversely, those who have not prepared to handle difficult situations are likely to be looking for work elsewhere.

Creating a Disaster Plan

The first step in preparing for emergencies is to do everything possible to avoid them. A "disaster" somewhere else doesn't have to mean an emergency for your systems. Rather than sending out a distress call, how much better to send the following sort of message to management: "You may have heard that a tornado ripped through downtown Salt Lake City last night—I just wanted you to know that our systems are all backed up, with full power, running smoothly and without interruption." The preparation that allows such a message begins with a **disaster plan**—a written document stating how you have prepared your systems and what you will do in the event that various problems occur. A disaster plan should cover a wide range of potential problems. Your plan may include guidelines regarding some of the following problems:

- A hard disk crash.
- An employee or intruder destroying data on the system or attempting to crash it.
- A fire in the server room.
- A natural disaster that wipes out all backups in the office, destroys the server, and blocks anyone from getting near the office.

A disaster plan should outline policies and procedures to guide the actions of the system administrator and other staff members in preparing for and reacting to many types of emergencies. Information in a disaster plan should include the following:

- Detailed system specifications for all hardware
- A list of software installed on the server
- Location of software masters, manuals, and licensing information for each application
- Information about the server's power supply
- Steps required to start the server using a boot or rescue disk (described later in this chapter)
- Steps required to reinstall critical applications and data archives for those applications
- Names and contact information for those to be informed of server emergencies
- Names and contact information for those who can be called to help in case of emergencies (including key technical support numbers for hardware and software vendors)

The exact steps outlined in the plan depend on the nature of your organization. For example, a company that provides credit card services to customers around the world could not even allow a natural disaster that wiped out an entire office to stop customers from accessing the network. In such a situation, millions of dollars are at stake; as a result millions should be spent in preparing systems that will prevent a problem in one area from blocking access to the entire system. A study done by Oracle, the largest database software company, estimated that for its key customers, every *hour* of system downtime resulted in a loss of $80,000 to $300,000.

On the other hand, a small company with a server or two is not subject to such losses, though it may be just as dependent on its computer systems. Small firms can take some simple precautions that will greatly help in nearly all situations. A critical difference is the amount of time that the organization is willing to be without data.

By performing all the tasks on the following checklist, you could get any office up and running again in 24 hours after most emergencies, at much less cost than keeping duplicate servers in cities around the country.

- Keep a replacement hard disk and possibly other parts (power supply, network cards) in storage.
- Back up the system each night to a rotating set of archive media (as described in Chapter 14).
- Take copies of the back-up media off-site once per week (to another office, a bank vault, or even someone's home in the case of a small company).

As you draft a disaster plan in consultation with management teams and others involved with the information systems at your organization, consider the monetary value of time. Can you calculate the dollar value of each minute that data is inaccessible to employees or customers? Many companies can; they should then spend accordingly to keep systems up and running.

Depending on your organization's requirements, you should consider all of the following components as potential points of failure, setting up redundant systems when possible:

- Hard disks
- Memory chips
- Network cards
- Power supplies (within a computer)
- Incoming AC power (wall socket)
- Software failure (kernel or application crash)

Preventing Downtime

The term **downtime** refers to occasions when an organization's computer systems cannot respond to requests for information. For example, a Web server that doesn't respond to browser requests, a database server that can't respond to queries, or a print server that can't accept print jobs are all experiencing downtime. In computer parlance, these devices are down. It's important to remember that a computer can be down without being powered off. If software or hardware problems prevent a computer from responding as intended, it is effectively down—because users can't perform tasks as they need to. The rest of this chapter describes techniques that you can use to protect your Linux system from excessive downtime. Before you can successfully prevent downtime, you need to understand a few key concepts:

- **Redundancy:** A state of readiness in which a duplicate component can take over if the primary component fails. A completely redundant system is a replicated system. Most computer systems are not completely redundant. Because of cost

considerations, only those components that are most likely to fail or that cause the greatest problems if they fail are supported by redundant systems.

- **Fault tolerance:** The ability of a system to deal effectively with problems (also known as faults). Faults can include hardware failure, power failure, stolen data, and many others. The more fault tolerant a system is, the more unexpected problems it can handle without going down.
- **Points of failure:** Weakness or vulnerabilities in the information system. Looking for the possible points of failure is a good way to strengthen the fault tolerance of your system. Work to eliminate any **single point of failure**—a single highly vulnerable part of a system whose failure will bring down the entire system. If a single (or most dangerous) point of failure cannot be eliminated entirely, try to make that part of the system redundant.

Understanding High Availability

Fault tolerant computer systems that run continuously are said to provide **high availability**. A high availability computer system is able to serve data as much of the time as possible. The availability of a system may be measured as a percentage. For example, a system may have a statistical availability (uptime) of 99.99%. Most standard Pentium-class systems running Linux would probably have an availability of 95% or higher, depending on the environment in which they are running. (For example, if the municipal power supply was subject to frequent outages, the availability might be lower.) But improving system availability becomes increasingly difficult as you approach 100%. To achieve 100%, all possible causes of downtime must be eliminated through redundant systems, elimination of points of failure, and so forth.

It's important to keep in mind that while achieving 100% uptime is an admirable goal, it is only a theoretical goal. All systems will necessarily experience some degree of downtime. To understand the difficulty implied in attempting to achieve 100% availability, think of a graph in which a curve approaches a line (an asymptote). The curve of availability never actually touches the line of 100%, but as it gets closer and closer, the costs rise to infinity. Thus, companies are forced to set more practical availability goals, based on the amount of downtime the organization can realistically tolerate. One way a company can accommodate unavoidable downtime is to schedule it in the form of maintenance or upgrades. A network that is carefully maintained and upgraded in this way is far less likely to experience costly, unexpected downtime.

Downtime is generally measured as a percentage. For example, suppose your organization can tolerate one hour of downtime per year. In other words, out of the 8760 hours in a year, 1 can be allocated to downtime (scheduled, if all goes well). Dividing 1 by 8760 gives a value of 0.0001142. Multiplying by 100 gives the percentage of downtime: 0.01142%. Put another way, this organization requires 99.989% uptime. Once you have calculated this goal, you can more easily determine the amount of time and resources required to achieve it.

Within the realm of high availability systems, the term **resource groups** is used to refer to an application, the data that it requires to operate, and any other resources the application requires to complete its computing tasks. The term **high availability cluster** is used to refer to a group

of servers that have high availability features and are dedicated to handling a common set of tasks. Each high availability cluster is devoted to handling a set of tasks or resource groups. Several methods are used to control how the servers in a cluster respond to problems. The two most widely used of these methods are:

- One or more servers sit idly waiting to take over a resource group (a task) in the event that a main server fails. When the main server is restored to activity, the back-up server becomes idle again as the main server takes over. This method requires additional hardware equivalent to the main server to provide redundancy, which may be costly.
- Each server in a cluster may handle various tasks. If one server goes down, the other servers take over the resource groups of that server. This doesn't require costly hardware to sit idly waiting for a failure, but performance may be seriously degraded if all servers are already busy when one fails.

Linux supports some high availability features. These include UPS devices and both hardware and software RAID (these subjects are discussed later in this chapter). However, Linux does not support some high availability features, such as software control of high availability clusters and mainstream applications that are aware of high availability features. Many people in the Linux community are working on getting Linux to support additional software tools and hardware components. In the future these will allow Linux to compete better with UNIX vendors that have been working on high availability systems for decades. These systems, from vendors like Oracle, IBM, and Hewlett-Packard, may cost millions of dollars. They include all of the redundancy features described in this chapter, as well as others that Linux does not yet support. By increasing its high availability features, Linux can be used in more business critical computing environments.

Creating Rescue Disks

One of the most common system failures is a hard disk problem. This can take the form of a complete crash, in which case you may have to restore all of the data to a new hard disk from a back-up archive. In other cases, part of the disk may be corrupted so that the system cannot boot correctly. Sometimes a hard disk problem can even be caused by something that the system administrator has done to the system. In the case of many hard drive failures, you cannot boot the system normally from the hard disk; you must boot the Linux system from a floppy disk before you can access the hard disk to try to solve the problem. This disk is called a rescue disk.

A **rescue disk** is a floppy disk that includes the files needed to run a minimal version of Linux without using the hard disks on your system. After booting the system from such a disk, you can mount the hard disks of your system (as described in Chapter 8) and edit files (including the LILO boot information) to correct problems. A rescue disk is only one type of floppy disk you may use with your Linux system:

- A rescue disk allows you to run Linux without relying on data stored on your system's hard disk. It contains a text editor such as `vi`, a few command-line utilities such as `ls` and `cat`, and a few utilities for managing hard disks such as `fdisk`. A rescue disk is used only for emergencies.

- A **boot disk** is used to launch the operating system stored on your hard disk. It is used in normal operating situations to start the Linux system.
- An **install disk** is used when you first install Linux. Its contents are similar to a rescue disk, but it is not designed to assist you in recovering damaged data.
- A **root disk** is used in conjunction with an install disk during some Linux installations (depending on the Linux distribution you are using).

Some system administrators always boot their servers from a boot disk. This takes more time than booting from a hard drive, but because a server is rebooted only rarely, speed of booting is not really a concern. By using a boot disk, the system administrator can control how the system boots and easily maintain a backup of the boot disk in case of problems. A rescue disk is used when the boot disk or hard disk fails.

Ideally, you would create a rescue disk for your system when installing Linux. This guarantees that the rescue disk and the information on the computer's hard disk are compatible, which may help as you attempt to fix system problems.

As an example, the Red Hat 6.0 Linux CD provides copies (or images) of the two disks needed to start the installation program: a boot disk and a root disk (the root disk contains a minimal root file system). Once you create these disks, you can use them to start your Red Hat Linux system at any time. The images for these disks are located in the `images/` subdirectory of the installation CD. You can copy these images to a floppy disk using the `rawrite` utility in DOS or Windows (which is also on the Red Hat CD). Alternatively, you can use the Linux `dd` utility by combining the `dd` command with the name of the image file (`rescue`, `boot`, or `root`). For example:

```
dd if=rescue.img of=/dev/fd0 bs=72k
```

To start a Red Hat 6.0 system in emergencies, you must first boot the system using a boot disk, then insert the rescue disk to begin the process of dealing with problems on the system's hard disks. Red Hat refers to this process as entering **rescue mode**. You can also create a boot disk in Red Hat 6.0 using the `mkbootdisk` command.

Other Linux distributions require similar techniques for creating a boot disk or rescue disk. Many installation programs also provide multiple virtual consoles that you can use to diagnose or repair problems after booting the system from an installation disk. By switching from one virtual console to another, you may be able to mount a hard disk and diagnose or repair a problem. Press the Alt key and then a function key (F1 to F6) to switch between virtual consoles. When using an installation disk rather than a rescue disk for this task, keep in mind that the installation disk may not have the tools you need to diagnose and fix a problem.

Maintaining Software Masters

As you will learn in Chapter 14, most back-up strategies focus on backing up user data—that is, the information created by your organization, for use only within your organization. System administrators are often careless about backing up the software tools and operating system files

used to create and work with that data because they think these items can be purchased in any computer store. However, this is not always the case. Consider the following problems that can occur after a system failure, when you attempt to reacquire your company's software tools:

- The company that created the software may be out of business. Note that this is most likely to happen when the software is specific to one industry, such as a video store software package, chemical plant software, or medical office software.
- The particular version used by the company may no longer be available from the vendor. A costly upgrade would then be required, with new training for all employees.
- The vendor may not be able to ship the replacement application software immediately. For instance, if your system goes down on a Friday, the vendor probably wouldn't be able to ship the software until Monday, for delivery on Tuesday. Thus, the process of reconstructing the system data from backups could not begin until five days after the initial failure.

In all of these situations, a system administrator would be better served if he or she had carefully and securely stored the **software masters** (such as the installation CD) originally provided by the vendor. Unfortunately, software masters are often discarded along with the unread user manuals as soon as the products are installed. Instead, they should be stored along with backups of other system data and mentioned prominently in your disaster plan. When using complex applications such as client/server databases or customized business software, application data sometimes cannot be restored properly until the application is first installed. If you can immediately locate software masters for all important applications used on your server, you can then rebuild a system as follows:

- Rebuild or repair hardware components such as hard disks.
- Install the operating system and applications used by the company.
- Restore user data files.

If you can afford the additional capacity, it's a good idea to back up the operating system and applications as well as user data. This safeguards your custom configurations as well as the applications themselves. But keep track of the software masters and documentation nevertheless. You may need them to retrieve license information or activation keys for your applications.

MANAGING THE COMPUTER'S POWER SUPPLY

The power supply is one of the easier components to protect within a Linux server. This component converts the **AC power** from a wall socket to the low-voltage DC power used

by computer chips, disk drives, and other accessories. As shown in Figure 9-1, the **power supply** is usually the single largest component inside your computer. It rarely fails if it is properly cooled by an internal fan or room air conditioner. For occasions when the power supply does fail, two possible remedies can be part of your disaster planning:

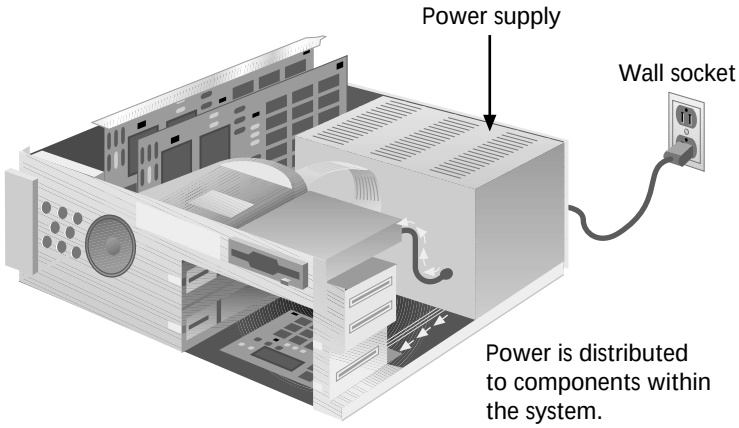

Figure 9-1 The power supply component inside a computer

- Keep a second power supply on hand. If the power supply inside a computer fails, you should be able to replace the part and have the system running again within an hour. Spare power supplies are not overly expensive.
- Purchase a server with a built-in back-up power supply. This is similar to having an uninterruptible power supply (described in the next section) within the computer. It allows you to schedule a time to shut the server down when it is not being used so you can replace the failed power supply component. Servers with dual power supplies are generally quite expensive because of the control circuitry required to switch to the back-up supply in case of a failure.

A power supply is a single component in your computer. Don't ever try to disassemble and repair it, or you risk receiving a powerful electric shock. Replace it as a complete unit. Opening a power supply unit is very dangerous. A power supply contains components such as capacitors that can hold a large electric charge even when the power to the system is shut off.

Connecting a UPS to a Linux System

In most parts of the world, failure of or an unexpected irregularity in the AC power (the wall socket power) is much more likely than a failure of the computer's power supply. You can ameliorate the effects of irregular power by using devices known as surge suppressors. **Surge suppressors** provide a regular power signal to the computer's power supply. These devices remove voltage spikes, brownouts (moments of reduced voltage), and other irregularities (such as changes in the frequency of the signal—normally 60 cycles per second) that don't affect your

refrigerator or office lights, but can damage sensitive electronic equipment. Figure 9-2 illustrates how a surge suppressor removes irregularities so that the computer's power supply can rely on a consistent level of incoming power.

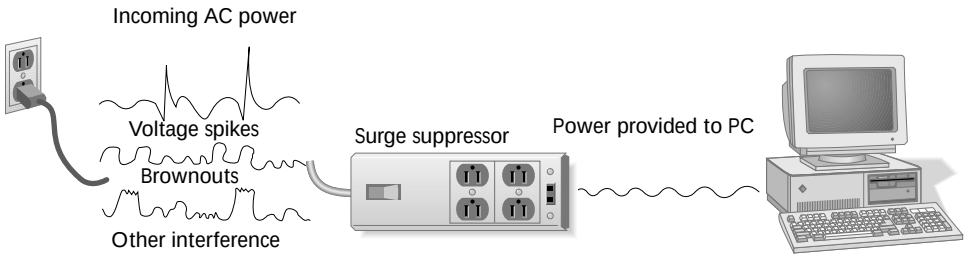

Figure 9-2 A surge suppressor removes power irregularities

When the power actually fails, however, a surge suppressor doesn't help. For these cases, an **uninterruptible power supply (UPS)** is needed. A UPS contains batteries that are continually charged by the main AC power when it is active. During a power outage, the batteries in the UPS take over, delivering power to the computer as if the outage had not occurred. Figure 9-3 shows a computer connected to a UPS.

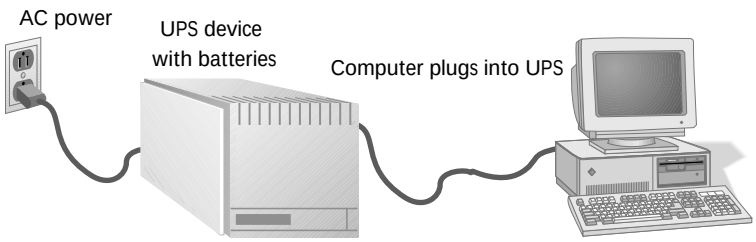

Figure 9-3 Computer connected to a UPS

A UPS also includes surge suppressor features. Thus when the main AC power is active, the power reaching the computer is filtered by the UPS; a separate surge suppressor is not necessary.

Because batteries are expensive (and heavy), a low-cost UPS typically provides only 5 to 30 minutes of battery power to a computer. Systems that require high availability may have a UPS costing thousands of dollars, that provides power for several days. Such high-end UPS systems may be integrated with gas-powered generators that recharge the UPS batteries. In most situations, however, a UPS is intended to provide power during momentary power outages, which generally last less than five minutes. In the event of a long-term outage due to weather or utility company problems, a UPS allows computer systems to be shut down gracefully. This is especially important for a Linux computer system, in which data is cached in memory rather than being written immediately to hard disk. If the system is not shut

down properly, such data will be lost. What's more, when the Linux system is turned on again, the system may require a few minutes (or several hours) to check the integrity of the hard disk files and perform other checks. A UPS prevents these problems by giving the Linux system administrator time to shut down Linux properly, inform users that the server will be shut down, and exit system services without any loss of data.

Automating Linux Shutdown

Although a UPS allows time for a system administrator to shut down a Linux system gracefully, the system administrator may not be present when the power outage occurs. For this reason, UPS devices typically have a serial communications port (similar to the connection used by a printer or external modem) that signals the Linux system in the event of a power outage. The Linux system then executes simple scripts to determine what actions to take. UPS devices can generally recognize three events, as described in Table 9-1.

Table 9-1 Events Tracked by a UPS Device via a Serial Port

Event recognized by UPS	Linux system response	Notes
Power failure	Begin a system shutdown	The shutdown process may take 5 to 30 minutes, depending on the capacity of the UPS device and the type of work and number of users supported by the Linux server.
Power is out and the UPS battery is low	Shut down quickly	In this situation, Linux usually executes an immediate `halt` command to write cached data to files and unmount file systems before the UPS battery fails. Users logged in to the system may have very little warning about closing their files, but typically no data is lost.
Power has been restored	Cancel any pending shutdown	When a nonemergency shutdown (taking 5 to 30 minutes) is in process, the shutdown can be canceled before any services are shut down.

Each item in Table 9-1 corresponds to a signal number (similar to those used with the `kill` command, as described in Chapter 8). When a UPS is connected to a computer running Linux, the system administrator also installs a software daemon to monitor the serial cable that the UPS uses to communicate with the computer. Figure 9-4 shows a UPS connected to a Linux system via a serial cable.

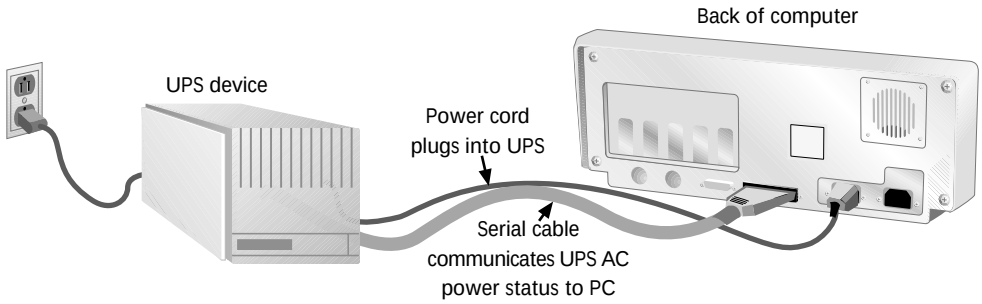

Figure 9-4 Serial connection between UPS and Linux system

Many different programs are available to monitor a UPS. One popular UPS manufacturer, American Power Conversion (APC), has Linux-specific programs designed to provide access to special control features of its UPS devices. Because most UPS devices employ common communications systems, however, the standard Linux UPS daemon, known as `powerd`, can be used with any UPS. The `powerd` software package may be included on your Linux CD-ROM. You can also acquire the `powerd` software package from Linux FTP sites such as *metalab.unc.edu* in the `/pub/linux/system/ups` subdirectory. The package on this FTP site is in `.tar.gz` format, so the command to prepare it on your system would look like this:


```
tar xvzf powerd-2.0.tar.gz
```

After installing the `powerd` software package, the system administrator must activate it by performing the following three steps:

1. Add `powerd` to the start-up scripts so that it is started each time the Linux system boots. (This step is described in the following section.)
2. Set up the `powerd` configuration file. This file controls how the `powerd` program monitors the serial communications line connected to your UPS.
3. Check the `/etc/inittab` file to see what actions Linux will take based on the instructions sent from the `powerd` program.

Autostarting the `powerd` Daemon

Before the `powerd` daemon can start automatically each time you boot Linux, you must add it to the system initialization scripts described in this section. The exact method required to accomplish this depends on your version of Linux, though the variations between Linux versions are small. This section describes a typical Linux configuration.

The `powerd` software package includes a script used to start the `powerd` daemon. The `powerd` installation process copies this script to the `/etc/rc.d/init.d` directory. In some cases you may need to manually copy the script to the correct directory. For example, if you stored the `powerd` package in the `/tmp` directory, the command to copy the script would look like this:

```
cp /tmp/powerd-2.0/powerd /etc/rc.d/init.d/
```

Once this script is installed in the correct directory, you can execute it (and start the `powerd` daemon) with the following command:

```
/etc/rc.d/init.d/powerd start
```

To stop the `powerd` daemon, use this command:

```
/etc/rc.d/init.d/powerd stop
```

This `powerd` start-up script should also be added to your system initialization directories so that `powerd` starts automatically each time you boot your Linux system.

To have Linux run the `powerd` start-up script automatically when the system is booted, add a pointer from a run level directory to the `powerd` start-up script.

The run level directories define which services Linux starts when using a given run level. Run levels 3 and 5 are most often used for standard systems. The run level directories are located in `/etc/rc.d`. For example, the directory used to initiate services for run level 3 is `/etc/rc.d/rc3.d`.

To add the `powerd` script to the run level 3 directory, create a link, or pointer, in that directory that refers to the start-up script. Each link in this directory begins with a letter `S` followed by a two-digit number. The numbers indicate the order in which services will be started.

The `powerd` daemon should be started before most other system services, so a value of `03` is a good choice. To create this link, use the following command:

```
ln -s /etc/rc.d/init.d/powerd /etc/rc.d/rc3.d/S03powerd
```

A similar link (using a filename beginning with `K` for `kill`) is used to indicate that the `powerd` daemon should be shut down when run level 0 (halt) or run level 6 (reboot) is initiated by the system administrator. These two commands create links within the run level 0 and run level 6 directories that point to the `powerd` script. Services are shut down in the reverse order of how they are started up, so the number used for these links is 90. The `powerd` daemon will be shut down after almost all other system services are shut down. To create this link, use the following command:

```
ln -s /etc/rc.d/init.d/powerd /etc/rc.d/rc0.d/K90powerd
ln -s /etc/rc.d/init.d/powerd /etc/rc.d/rc6.d/K90powerd
```

The exact numbers used on your Linux system, the directories where these files are contained, and the specific daemons started in each run level will vary depending on your Linux distribution. The numbers that you select are somewhat arbitrary. They should simply be between the numbers used for other services on the system. You can review these other services by examining the contents of the run level directories (such as `/etc/rc.d/rc3.d`).

Configuring the `powerd` Daemon

The configuration file used by the `powerd` daemon, named `powerd.conf`, is located in the `/etc` directory. By default, most of the options in this file begin with a hash mark (`#`), indicating that they are commented out (marked as comments and thus inactive). By reviewing the

comments, you can learn how each feature is used. Then you can select the features you want to use and activate them by deleting the hash mark. Although the `powerd` daemon supports many features, the configuration available to you depends on your UPS device. The remainder of this section describes a generic configuration for signaling a power failure.

In order to be a successful system administrator, you don't need to understand everything about serial communications, but you will need to learn about how your UPS device signals a power failure or low battery condition. Specifically, to understand how the UPS informs the `powerd` daemon of a failure, you must understand the structure of a serial connection. A standard serial cable contains either 9 or 25 pins, as shown in Figure 9-5. These pins can be referred to by number, as indicated in the figure. A small number of these pins (wires inside the cable) are named for the functions they perform, such as carrying a bit of data or signaling that a bit of data was received successfully. (The exact names are abbreviations of the pin's function. For example, the pin that indicates "ready to send data" is known by the label RTS.)

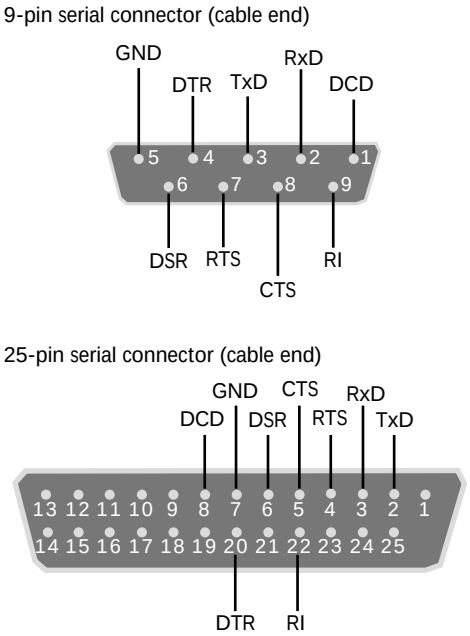

Figure 9-5 Standard serial cable connections

As one example, a Blackout Buster UPS from PK Electronics uses pin number 1 on a 9-pin cable to signal a low battery condition. When the UPS sets the voltage on pin number 8 to low, this indicates that the main AC power has failed. By contrast, a non-low voltage on that pin indicates that the power outlet is working. On any 9-pin serial cable, pins 1 and 8 are referred to, respectively, by the names DCD and CTS. The `/etc/powerd.conf` file contains information about the pin numbers and corresponding names for 9-pin and 25-pin cables, but you must learn the specifications of your UPS to determine which pins or names to use in configuring the `powerd` daemon. The technical support number or Web site of the UPS manufacturer generally has all of this information readily available.

UPS vendors' software packages are usually designed to provide automatic shutdown for NetWare or Windows servers. Note that this software will not work with your Linux system. Very few UPS vendors currently support Linux. American Power Conversion (see *http://www.apcc.com*) is one exception.

The `powerd` program can be configured by making a few changes to the default configuration file. Within the `/etc/powerd.conf` file, update the following lines using a text editor of your choice:

1. Look for the line containing the word `disabled`. This line indicates that the `powerd` program should not actively monitor the serial port. By default, the `powerd` configuration is disabled to prevent it from listening to a serial port before it has been configured properly. An improperly configured `powerd` daemon might listen to the wrong serial port and attempt to shut down the computer, even though the power supply is fine. If necessary, add a hash mark (#) to the beginning of this line to make it a comment.

2. Use the search feature of your text editor to locate the phrase `debug yes` in the `powerd.conf` file. This statement indicates that the `powerd` program should not run in the background but should stay active on the command line after being started. This allows you to immediately end the `powerd` program by pressing Ctrl+C as you test that `powerd` works correctly. Change the line `debug yes` to `debug no`. You may want to do this step after testing the UPS configuration as described in the next section, but you should not reboot your system unless you have set this line to `debug no`. Otherwise, your system may pause while waiting for a response on the serial port.

3. The section labeled `serialline` includes three lines of text that define how the `powerd` program will monitor the serial port. The three lines define which serial port to monitor, the wire within the port to monitor, and the status of the line to look for (watch for either high voltage or low voltage). Uncomment these lines and set the correct values that you have determined by reviewing the documentation for your UPS device. For example:

```
serialline        /dev/ttyS1
     monitor           DCD
     failwhen          low
```

   These values indicate that the second serial port (`/dev/ttyS1`) should be monitored, that the DCD line within the port should be monitored, and that a failure signal should be sent when the DCD line goes to low voltage.

4. Save your changes to the `/etc/powerd.conf` file and restart the `powerd` daemon using the following two commands so that the configuration file is reloaded by `powerd`:

```
/etc/rc.d/init.d/powerd stop
/etc/rc.d/init.d/powerd start
```

You may see device names (such as `/dev/cua0` or `/dev/cua1`) within the `powerd.conf` configuration file. Formerly, the serial port devices on a Linux computer were known by the names `/dev/cua0` for the first port, `/dev/cua1` for the second port, and so forth. The preferred names for serial port devices in Linux have been changed to `/dev/ttyS0` for the first port, `/dev/ttyS1` for the second port, and so forth. You may still see the `/dev/cua`*x* device names—and they will still work on most Linux systems—but try to use the newer `/dev/ttyS`*x* device names where you have a choice.

Although a UPS has only one serial communications cable attached to it, every UPS has several power outlets; thus several Linux servers can be plugged into one UPS. Each Linux server that is powered by an outlet on the UPS should run a copy of `powerd`, even though the serial cable is not connected to each of the Linux servers. Thanks to additional features of the `powerd` daemon, it can use a network connection to inform a copy of `powerd` running on other Linux servers of a power outage. Thus, `powerd` on those other servers can watch for a message on the network from another copy of `powerd`. When this message is received, each copy of `powerd` can shut down the server on which it is running. The `remoteserver` line in the `/etc/powerd.conf` file is used to configure this remote monitoring feature. Figure 9-6 shows how several servers can be plugged into one UPS device. A single serial cable allows the UPS to inform one server of a power outage. That server then informs other servers via the network. (All of these messages are communicated between copies of the `powerd` daemon.)

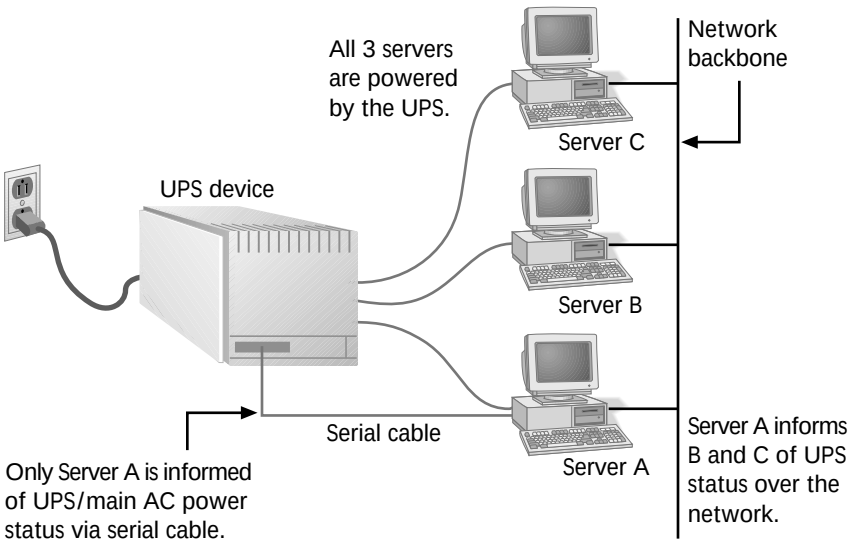

Figure 9-6 Multiple servers connected to one UPS with remote server shutdown

Responding to a Power Outage

The `powerd` daemon doesn't shut down your Linux system. Instead, it watches the serial port for information from the UPS device. When it receives information indicating the

power is out, the UPS battery is low, or the power has been restored, it communicates a signal to the `init` program. The **`init`** program is a control process that starts the first processes on Linux, such as the login screens.

The `init` program is configured by a file called `/etc/inittab`. When the `powerd` daemon sends a signal to the `init` daemon, `init` checks the `/etc/inittab` configuration file to determine what action to take. The file `/etc/inittab` normally includes two or possibly three lines indicating how Linux should respond when a power outage occurs, the UPS battery is low, or the power is restored. The exact content of these lines (that is, the exact actions they specify) depends on your Linux vendor and version, but these lines taken from Red Hat 6.0 Linux are good examples:

```
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System
  Shutting Down"
```

```
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown
  Cancelled"
```

The first part of the line (`pf` or `pr` in these two example lines) is a label chosen by Red Hat. The second part, `powerfail` and `powerokwait`, indicates to the `init` daemon what is being configured: either power failure or power restored. Only two possible signals are represented here. (Testing for a low battery condition is not part of the `/etc/inittab` configuration in Red Hat 6.0.) In another distribution, the configuration line for a low battery would look something like this:

```
p1:powerfailnow:/sbin/shutdown —h now "Battery Low…"
```

All three of these signals make use of the `shutdown` command. In each of the three relevant lines in `/etc/inittab` (monitoring signals for power failure, UPS battery low, and power restored), different parameters are included after the `shutdown` command to determine how `shutdown` operates. Each of the three commands is described in the following list. In each case, the message in quotation marks is part of the `shutdown` command and appears immediately on the screen of every user who is logged in to the Linux system.

- `shutdown -f -h +2 "Power Failure; System Shutting Down"`: Initiates a timed shutdown that warns all users on the system that the server will be halted (shut down and not automatically rebooted, using the `-h` option) in two minutes. No system services are actually shut down until the end of the two minutes. If your UPS can supply power for a longer time, you could change the `+2` parameter to another number—for example, to `+10` for 10 minutes.
- `shutdown -c "Power Restored; Shutdown Cancelled"`: Cancels an existing shutdown. This command is useful after a timed shutdown has been announced to users but before the time has expired. If power is restored, there is no need to shut down the system, so another announcement is sent to all users that the shutdown has been canceled and they can continue working normally. This would be used if a short power outage causes the `init` program to announce a shutdown in 2 to 10 minutes (or more, depending on your UPS), but the power was restored after a short time.

- `shutdown -h now "Battery Low"`: Initiates an immediate shutdown (halt and do not automatically reboot, using the `-h` parameter with `now`), disconnecting any users that are connected via the network and ending all applications. When the UPS battery is failing, the time left before the power is cut to the computer cannot be determined, so this rather impolite command is used. Though it is unpleasant for users because it ends their Linux session without warning, it is better to preserve the file system integrity than to risk losing data because the battery fails before a shutdown has been completed. As a rule, you can avoid low battery conditions by careful planning. Ideally, your UPS device would be adequate for your Linux system, which means this command would never be necessary.

You can update the information in the `/etc/inittab` file so that the `init` daemon executes any command that meets the needs of your system. The `shutdown` command is effective for unmounting file systems and preventing data loss, but you might also want to create a script (see Chapter 12) that completes a set of tasks related to shutting down the system. These tasks might include:

- Writing an incident to a log file
- Sending an e-mail message to another system
- Signaling a back-up server to take over a critical task
- Writing information to a networked hard disk so it can still be accessed
- Starting the `shutdown` command to halt the system

Checking File System Integrity

Maintaining the integrity of file systems is a key part of protecting data on your Linux systems. In order to maintain the file system, it's useful to understand some basics about the information stored on a file system and the arrangement of that information. The definitions in the following paragraphs apply to all Linux `ext2` file systems.

The **superblock** is a set of information about the entire file system. It includes items such as the following:

- Maximum number of times the file system can be mounted between complete file system checks (using the `fsck` utility described in the next section)
- Size of the file system in blocks (the standard block size is 1024 bytes)
- Number of free blocks on the file system
- Number of reserved blocks (reserved blocks can only be used by the root user or another user or group that the root user specifies)
- Amount of time since a complete integrity check was performed on the file system

If the superblock is corrupted, the entire file system is unusable. Copies of the superblock are stored in multiple locations across the hard disk during the formatting of an `ext2` file system to protect against a superblock problem rendering the system unusable. Some parameters stored

in the superblock, such as the maximum number of times the file system can be mounted between integrity checks, can be configured during formatting or reset using the `tune2fs` utility described later, in "Tuning a Linux File System." Some features of the file system, such as the size of a data block, cannot be changed without reformatting the entire file system.

An **inode** (pronounced eye-node) is a file information record. A **file record** contains a filename and the inode that holds information needed to access the file. The file record is an indirect pointer to the file's data: it points to an inode, which in turn points to the data in a file. A **directory record** is a file containing a list of files and corresponding inode numbers used to access data in those files.

Each inode contains all of the information about a file except the file's name. This information includes the following:

- The file access permissions
- The owner and group of the file (noted as a user ID and group ID number)
- The time and date of the last access, last modification, and file creation
- Number of blocks used by the file on disk and the precise size of the file in bytes
- A pointer to the blocks of data in which the file's contents are stored

Because of the way inodes are arranged on an `ext2` file system, a file system can have only a limited number of inodes and thus a limited number of files. The number is large, and you're never likely to run out of inodes for creating new files. But you will notice as you use the `fsck` utility in the next section that the utility displays messages indicating the maximum number of files that the file system can hold. For example, the utility output reading `43686/247808 files` indicates that of 247,808 possible files on this 1 GB hard disk, 43,686 have been created thus far.

The brief explanation of inodes provided in the previous paragraph should help you understand how links are used in the Linux file system. A **link** allows two or more file records to refer to the same physical data stored in a file system. Among other things, a link makes it possible for two users to have two different filenames listed in their home directories, but actually be working with the same data when they edit those two files. Links are of two types: symbolic and hard. A **symbolic link** is a file record that includes a path and filename, but not an inode. When a user refers to a symbolic link, Linux looks at the path and filename mentioned in the file record—sort of like an internal URL pointing to another filename. The file record for that second path and filename includes an inode. That inode is used to access the file data for the symbolic link.

A **hard link** is a file record that includes a filename and inode, just like a regular file record. But a hard link refers to an inode that already has a file record pointing to it. The hard link is a second file record pointing directly to the same physical data. A single inode can have numerous file records (hard links) pointing to it. After a hard link is created, it is equal to the first file record that points to the same inode. If the first file record is deleted, the second file record (created as a hard link) is unaffected—it still refers to the same file data.

Both symbolic and hard links are used often in a Linux file system. Whenever you use the `ls -l` command, any symbolic links are indicated as extra filenames in the right column of the output. An example of a symbolic link in Red Hat Linux is the `view` command. This output shows how you can use the `ls -l` command to show the `view` command:

```
$ ls -l /bin/view
lrwxrwxrwx   1  root root 2   Aug 12 13:36  view  ->  vi
```

The arrow in the right column indicates that the file named `view` is a symbolic link to the file named `vi` (located in the same directory, because no pathname is included). The letter `l` in the far left column of the screen output also indicates that the file is a symbolic link.

The number in the second column from the left (in this case, `1`) indicates the number of file records that refer to the same inode as this file record.

The `zcat` filename (also in Red Hat Linux) is an example of a hard link. The `zcat` file record refers to an inode that two other file records also refer to. The `ls -l` command again shows this:

```
$ ls -l /bin/zcat
-rwx-r-xr-x    3  root root  50384    Mar 25 13:28 zcat
```


In this sample screen output, the number `3` in the second column from the left indicates that the file record holding the filename `zcat` refers to an inode that two other files also refer to (for a total of three). Figure 9-7 shows the difference between a symbolic link and a hard link. In this figure, a programming language named Perl is stored in a file called `perl5.003`, where the `5.003` indicates a precise version number. Other file records also point to the same information (inode) using different filenames. This allows users who might not know the precise version number to access the Perl programming language file.

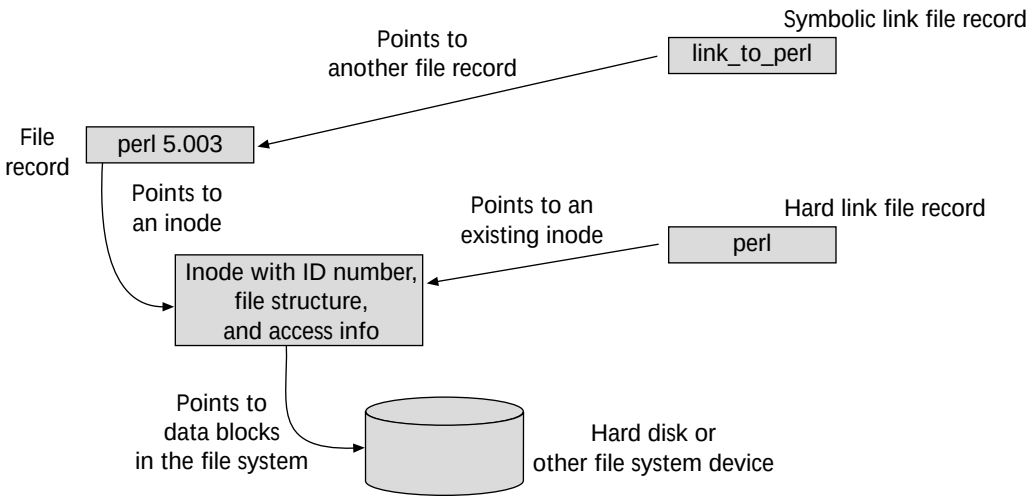

Figure 9-7 Hard link vs. symbolic link

To see the inode numbers associated with the filenames in a directory, use the `-i` option of the `ls` command. Only the first few lines of the output for this directory are shown in this listing:

```
$ ls -i /bin
22549 arch
22497 ash
22498 ash.static
22507 awk
22514 basename
22475 bash
22499 bsh
22495 cat
22477 chgrp
22478 chmod
22479 chown
22502 consolechars
22480 cp
22504 cpio
```

Using the `fsck` Utility

You can run the `fsck` utility at any time to check the integrity of a Linux file system. The `fsck` utility can quickly determine whether any file system problems are evident. If none are apparent, `fsck` runs very quickly. You can, however, force `fsck` to execute a full suite of file system tests even when the utility would otherwise exit quickly because no errors are apparent. This is called forcing the `fsck` utility.

Each time you boot Linux, the `fsck` program runs automatically to check all mounted file systems. Under normal circumstances, this automatic check takes only a few seconds. Two situations will cause the `fsck` utility to spend more time checking the file systems:

- Linux was not shut down gracefully (all system services were not shut down in an orderly way). Data was therefore not written correctly or fully to the file system.
- The file system has been mounted numerous times. After a certain number of mount operations, Linux forces the `fsck` utility to go through a complete check of the file system even if no problems are evident.

When the second condition occurs, Linux displays a message like the following as you boot the system:

```
Maximal mount count reached: check forced.
```

Just as Linux occasionally does a thorough (forced) file system check automatically, you can also use the `fsck` utility to manually check each of your Linux file systems. Such a complete check is wise before any disk-intensive operations, including the following:

- Backing up the system completely before a hardware upgrade
- Defragmenting the file system (as described in the next section)
- Attempting to repair data in a corrupt (damaged) file system

To force a complete check of a file system, use the `fsck` command with the `-f` option, followed by the name of the device on which the file system is located. The following example of the `fsck` command checks the file system located on the first partition of the second IDE hard disk:

You can only use the `fsck` command on file systems that are *not* mounted. Using it on a mounted file system will damage the file system.

```
fsck -f /dev/hdb1
```

You can also use the mount point in the directory structure. For example, if the first partition of the second IDE hard disk (`/dev/hdb1`) is mounted as `/database`, you could also use this command:

```
fsck -f /database
```

Multiple file systems can be checked at the same time by listing them. For example:

```
fsck -f /dev/hdb2 /dev/hdb3 /dev/hdc1
```


The `fsck` program runs all the file system checks at the same time, querying you as questions arise about file system information. If a file system you are checking is fine, you see only a brief output after running `fsck`, as follows:

```
Parallelizing fsck version 1.14 (9-Jan-1999)
e2fsck 1.14, 9-Jan-1999 for EXT2 FS 0.5b, 95/08/09
/dev/hdb2: clean, 43686/247808 files, 906478/987997 blocks.
```

Launching `fsck` with the `-f` option provides more detailed information. Sample output is shown here:

```
Parallelizing fsck version 1.14 (9-Jan-1999)
e2fsck 1.14, 9-Jan-1999 for EXT2 FS 0.5b, 95/08/09
Pass 1: Checking inodes, blocks and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/hdb2: clean, 43686/247808 files (0.3% non-contiguous),
  906478/987997 blocks.
```

Note that running a complete file system check using the `-f` option takes several minutes. A 1 GB Linux file system that is almost full of data may take three minutes to check. Larger file systems take considerably longer. What's more, using `fsck` to perform file system repairs on a Linux system with multiple or large file systems may take up to an hour to restart the Linux system. You can avoid time-consuming file system checks by shutting down Linux properly every time.

A proper system shutdown in Linux is often referred to as a "graceful shutdown." An ungraceful shutdown does not allow Linux to store information to files, stop system services, and cleanly unmount file systems.

The screen output of the `fsck` utility shown above contains five complete passes through the file system. The `fsck` utility will automatically correct any problems detected during each of the five passes. If an automatic correction is not possible, a query will appear on screen. These queries are simple questions asking you whether to save or delete unidentified pieces of information, or to provide some other piece of information that the file system requires. It is rare to have `fsck` ask a question during a file system check unless a serious disk problem occurred, such as the power to the Linux system being cut off during a complex disk operation.

When Linux is booted, the `fsck` utility is run with the `-a` option, which automatically corrects any minor errors. These minor errors might include such things as a missing timestamp on a file if the system was not shut down properly.

In some cases it may be necessary to indicate to the `fsck` utility the type of file system being checked. This is done with the `-t` option. For example:

```
fsck -t ext2 -f /dev/hdb2
```

Defragmenting a File System

All file systems are divided into units of storage called **blocks**. A standard block contains 1024 characters (bytes). All of the files stored in a file system are broken into pieces and stored in these file system blocks. On many operating systems, the information stored in a file system becomes fragmented. That is, the blocks that make up individual files are spread all over the surface of a hard disk or other device. Reading an entire file on a fragmented hard disk requires much more time and movement of the hard disk mechanism than reading nonfragmented files, which are stored in sequential sections of the hard disk. Windows NT is particularly noted for the need for frequent disk **defragmenting**. (Defragmenting is also called **disk optimization** because it optimizes access time to files and reduces wear on the hard disk.)

Fortunately, the `ext2` file system used by Linux is not subject to much file fragmentation. When new files are written to a Linux hard disk, the files are stored at locations spread across the entire hard disk. This allows space for each file to grow in size without becoming fragmented. In contrast, most file systems used by other operating systems begin storing a new file at the first available location on the hard disk. As new files are added, they fill up the space immediately next to existing files. As a result, if an existing file increases in size, the operating system cannot simply squeeze more information into a space adjacent to the file. Instead, it must allocate the necessary space from a different part of the hard disk. As more and more files become broken into pieces in this way, the disk becomes fragmented. Figure 9-8 illustrates the difference between an `ext2` file system and most other file systems in this regard.

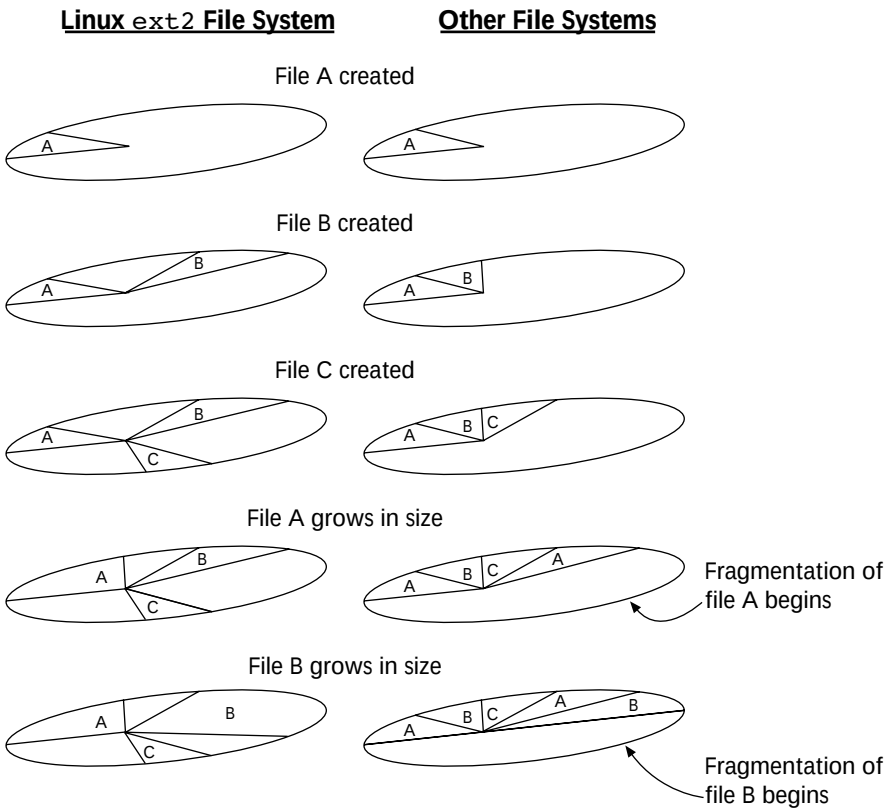

Figure 9-8 Creation of new files in ext2 and in other file systems

When you run the `fsck` utility (or watch it run automatically as Linux is booted), the screen displays the percentage of files on each file system that are fragmented. This percentage is normally less than 3%.

Hard disks that are becoming very full, however, are likely to experience more file fragmentation as space for new or expanded files becomes scarce. In this case, it can be helpful to defragment your Linux hard disk. Because this operation is rarely done in Linux, the utility used to defragment a Linux hard disk is not included with most Linux products. You can download it from the Internet using an FTP site such as *metalab.unc.edu*. Multiple utilities are provided in the defragment package to allow you to work with various types of file systems. The utility used to defragment a standard Linux `ext2` partition is called **`e2defrag`**.

When you need to defragment a Linux file system, begin by unmounting the file system. Then be certain to run `fsck` to check the hard disk's integrity. To complete these tasks on the `root` file system, you must boot Linux from a floppy disk so the root partition on the hard disk is not mounted.

Tuning a Linux File System

The **dumpe2fs** command provides many pages of information about a Linux ext2 file system. This command can be used on any Linux file system, whether mounted or not mounted. It provides details about the options set in the superblock of the file system and information about the data blocks. A paging program such as less should be used to view the output of dumpe2fs, as it runs to many pages. A sample command is shown here:

```
# /sbin/dumpe2fs /dev/hda3 | less
```

You can use the **tune2fs** utility to specify how an ext2 file system acts in certain situations. When you format an ext2 file system, the mke2fs command described in Chapter 8 uses default options for the items described in this section. You can add parameters to the mke2fs command to set those options as you format a new file system, or you can use the tune2fs command after the file system is formatted. Most of these options are stored as part of the superblock on the file system.

The tune2fs command should never be used on a file system that is mounted. Unmount the file system that you need to adjust, booting Linux from a floppy if the root file system must be tuned.

The tunable parameters for an ext2 file system affect the file system's stability and, to a lesser degree, its performance. Some configuration options of a file system, such as the block size, cannot be changed without reformatting the file system and using a parameter on the mke2fs command. The primary options available with the tune2fs command are described in the following paragraphs.

You can set the maximum number of times the file system can be mounted without having a complete integrity check using the fsck utility. A lower number will check for errors more often, but will also slow down system startup more often as the system is being checked. You can use the dumpe2fs command to see what the current maximum mount count value is set to. A sample command to set this value is shown here:

```
# /sbin/tune2fs -c 10 /dev/hda3
```

The number of times the file system has been mounted since the last time it was checked is recorded in the superblock. You can change this value using the -C option. By lowering or raising this number in relation to the maximum mount count parameter, you can control when the next forced automatic integrity check will occur.

You can determine the Linux system's response when a file system error is detected. The response is normally continue. However, you can choose to stop the Linux kernel when an error is detected, or remount the file system as read-only instead of read/write. (Note that mounting the file system as read-only may make the operating system unusable, although safe for examination and repairs.) A sample command is shown here:

```
# /sbin/tune2fs -e remount-ro /dev/hda3
```

Part of each Linux file system is set aside as reserved space. This space cannot be used by any regular system users for normal file storage. Instead, it is generally only available to the root

user when the hard disk becomes full so that `root` can continue to operate the system in order to free space for other users. If the entire `root` file system (including this reserved space) becomes full, the system is basically unusable.

This reserved space on a file system is normally 5% of the total file system space. However, you can configure this amount as a percentage of the total file system size or as a number of data blocks on the file system. You configure these options using the `-m` or `-r` option with the `tune2fs` command. The `root` user can always access this reserved space, but you can also allow another user ID number or group ID number to access it by using the `-g` and `-u` options of `tune2fs`. This might be useful, for example, if you have a file system dedicated to a database application and wish to reserve some space for the database administrator's group.

In addition to the `dumpe2fs` command, you can use the `-l` option of `tune2fs` to list the contents of the superblock. Because the `tune2fs` command sets options within the superblock, viewing the current settings with the `-l` option can be helpful.

Understanding Redundant Disk Systems

One of the most vulnerable parts of a computer system is the hard disk. It contains many moving parts, and it may be subjected to constant heavy use. Its health is crucial to the health of the entire system and to users' productivity, because the hard disk contains the data that the computer system was designed to provide. For these reasons, a great deal of effort has gone into making hard disks redundant—so they don't form a potential single point of failure in a system. Although redundant hard disks have long been available on mainframe systems costing millions of dollars, most users now rely on groups of hard disks, known as **redundant arrays of inexpensive disks**. More commonly, these groups of disks are called **RAID** subsystems or RAID arrays.

The idea behind RAID is this: rather than attempting to create a single expensive hard disk that never fails (a technological impossibility), it makes more sense to use a group, or **array**, of standard (inexpensive) hard disks. The assumption is that if one disk fails, the others can take over its duties until the failed disk is replaced. The safety of a RAID system lies in the extreme improbability of all the disks in the array failing at the same time. Of course, this assumes that high-quality components are used—an important assumption for RAID.

The management or control of a RAID system can be implemented in hardware or software. It can contain as many disks as necessary to reach the storage capacity needed. A RAID system does not have a specific number of hard disks, a specific storage capacity, or even a specific platform. RAID systems are used by all operating systems. Instead, the different forms, or levels, of RAID are distinguished by the techniques used to store data, as explained in the following sections.

Defining RAID Levels

RAID can be implemented in many forms, or levels. The levels differ in the amount of fault tolerance they provide and the cost of implementation. The next sections introduce the different RAID levels and define the terms and techniques associated with RAID.

Vendors occasionally differ in the features they associate with the different RAID levels. When reviewing RAID technologies or products, look for specific features rather than just a level number.

RAID-Linear

RAID-Linear is a technique that makes it possible to combine multiple physical devices into a single logical device. This allows one logical file system (as mounted in Linux) to span multiple disk drives or partitions. This is useful when you want a file system to be larger than a single hard disk. For example, suppose you need a huge file system for the `/home` directory. By storing this file system on multiple disk drives, you avoid buying one very large (and very expensive) hard disk. Instead, you can purchase multiple disks of a more common size. But the `/home` directory can still be managed as a single file system because of RAID-Linear.

By itself, this system doesn't provide any fault tolerance or data protection. RAID-Linear is not a true RAID level because it does not provide any fault tolerance, nor does it improve system performance as some RAID levels do. In fact, RAID-Linear reduces fault tolerance because if any one of the disks in a RAID-Linear array fails, the entire file system is unusable. RAID-Linear is illustrated in Figure 9-9.

RAID-0 (Striping)

RAID-0 makes use of a data storage technique known as striping. With **striping**, a single piece of data is divided into pieces and stored on more than one hard disk. This technique allows the system to access data faster, because two disk drives work together to gather parts of any requested information at the same time. The performance gain is increased if each hard disk uses a separate hard disk controller, such as having two SCSI cards; the multiple hard disk controllers prevent a bottleneck in communicating with the CPU via a single data channel. However, if either hard disk fails in a RAID-0 setup, the entire file system is unusable. Hence, using RAID-0 without other measures described in this chapter reduces fault tolerance significantly. Figure 9-10 illustrates the use of striping.

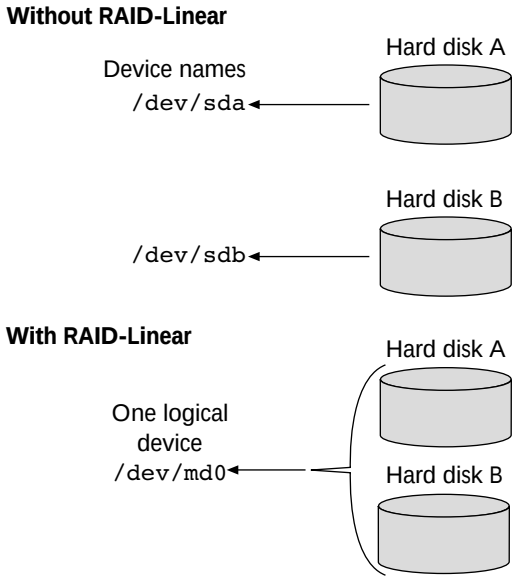

Figure 9-9 RAID-Linear

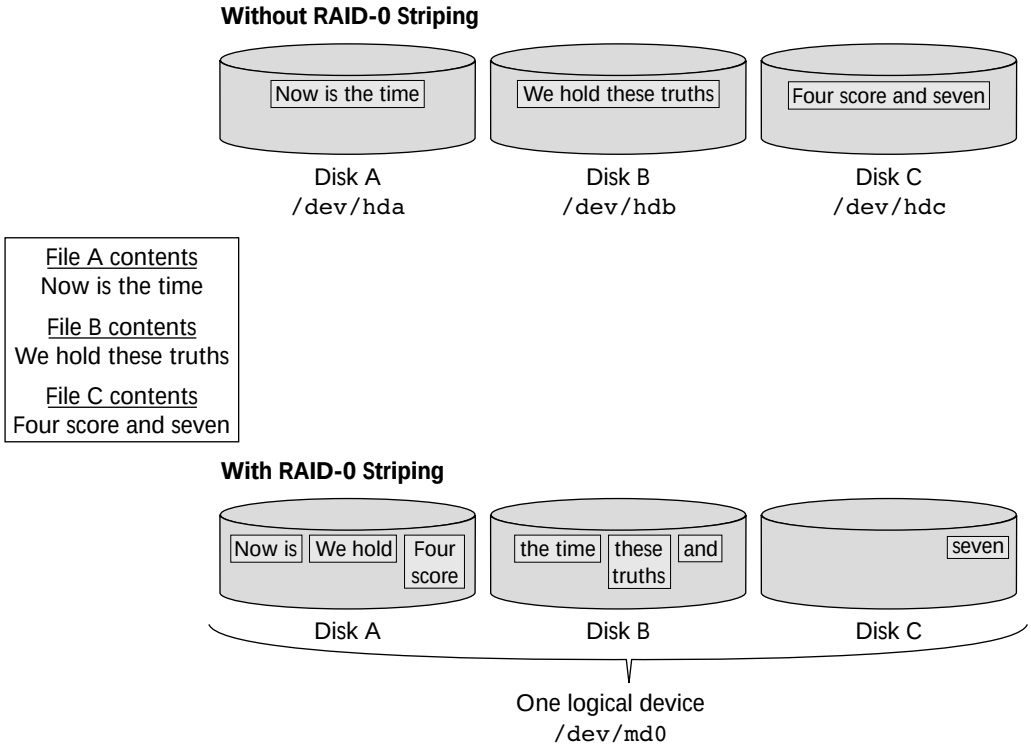

Figure 9-10 RAID-0, or disk striping

RAID-1 (Disk Mirroring and Duplexing)

RAID-1 mirrors data across multiple hard disks. The term **mirroring** is used to describe a system with two or more hard disks that contain identical information. Each time one hard disk is updated, the copy or copies are also updated. If one hard disk fails, the mirrored hard disks continue to respond to data requests without interruption. **Duplexing** is identical to mirroring except that the hard disks are on separate controller cards. The separate controller cards provide increased performance and reduced vulnerability to a hardware failure. Duplexed hard disks are sometimes referred to casually as mirrored disks, though in fact the use of multiple controller cards makes the two techniques distinct. Duplexing is the preferred technique when possible.

Mirrored and duplexed hard disks provide both increased performance (because multiple hard disks respond to data requests at the same time) and increased fault tolerance (because if one hard disk fails, the duplicate disks continue to work without interruption). Mirroring or duplexing disks does have two downsides. First, the time required to write files to a hard disk is increased because data must be written to each disk instead of just one. Second, these techniques require extra hard disks that are used only for mirroring—they don't provide extra storage capacity. For example, if you want to mirror a 10 GB hard disk, you must pay for 20 GB of hard disk space. With the low price of hard disks today, this is generally a very good investment, though it can be a limiting factor when planning a system that includes large amounts of storage (hundreds or thousands of GB of data). Figure 9-11 shows a mirrored hard disk system.

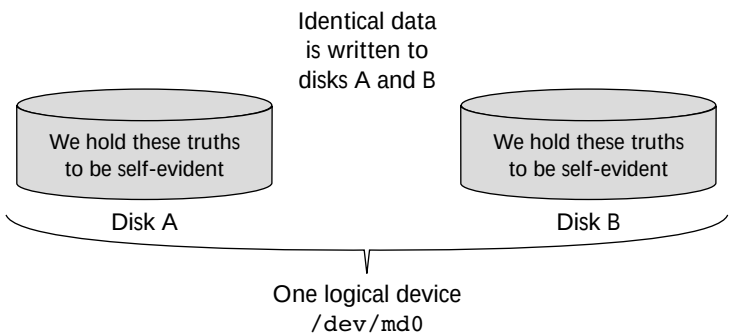

Figure 9-11 RAID-1 with mirrored hard disks

RAID-3 (Striping with Parity)

RAID-0 (striping) improves performance but makes a system more vulnerable to failure. **RAID-3** combines the performance advantages of striping data across multiple hard disks but provides additional protection against the failure of one of the hard disks in the form of parity. The term **parity** refers to a technique that allows corrupted data to be reconstructed using

an extra piece of information that is created as the data is stored. The parity information provides redundancy to the piece of data. In RAID-3, this extra information regarding how files are divided up is stored in a **parity stripe**. If one of the hard disks fails, the system can use the parity information to reconstruct the data stored on that disk. Figure 9-12 illustrates a RAID-3 system.

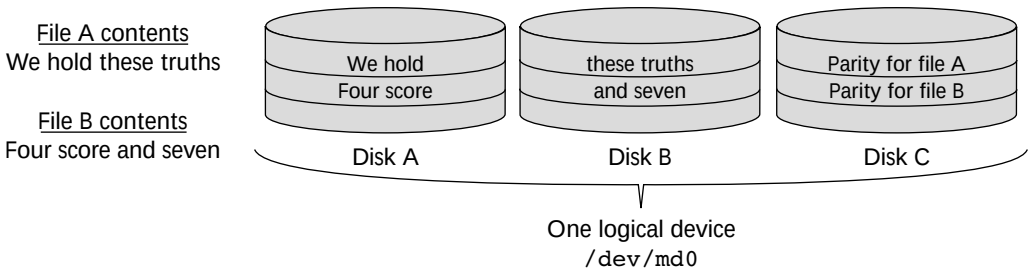

Figure 9-12 RAID-3, striping with parity

RAID-3 provides good fault tolerance because of the parity stripe. If a hard disk fails, data is still usable; when the disk is replaced, the data is rebuilt without any intervention by the system administrator. RAID-3 also improves performance on systems that read a lot from the disk. All the hard disks in a RAID-3 array can work in parallel to respond to multiple requests at the same time. Instead of waiting for a single disk, the chances are good that a hard disk that is not busy can immediately begin to service a read request. Also, unlike disk mirroring, with RAID-3 only one hard disk (the one containing the parity information) is unavailable for data storage. For example, to have 80 GB of storage space in a RAID-3 system, you only need to purchase 90 GB of hard disks. The write performance of RAID-3 does suffer because the parity information must be computed and several hard disks must store information for each write operation.

Although RAID-2 and RAID-4 are listed in some definitions of RAID, they are not used in production systems and so are not presented here. These two RAID levels simply use different combinations of the technologies implemented by other RAID levels.

RAID-5 (Striping and Parity)

RAID-5 is similar to RAID-3 except that with RAID-5 both the parity information and the stored data are striped across multiple hard disks. This has the advantage of making read performance better, but it makes write performance even worse than RAID-3. As with RAID-3, if a hard disk fails, the parity information allows the information to be reconstructed once the disk is replaced. In the meantime, data remains available.

Figure 9-13 illustrates how RAID-5 spreads information across multiple hard disks. Many vendors who sell RAID-5 hardware systems (see the next section) use built-in **write caching** to store new information in memory until it can be written to the multiple hard disks without degrading performance overall.

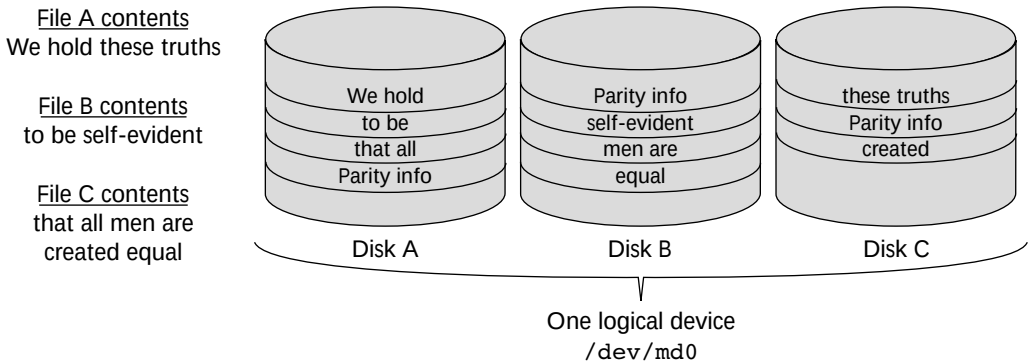

Figure 9-13 RAID-5, parity and striping over multiple disks

Using Hardware-Based RAID

A RAID system can be implemented as a separate device that connects to your Linux computer like an external printer or CD-ROM drive. This type of RAID is known as **hardware-based RAID** because the control and management of the disk array relies on a hardware system (called a **RAID subsystem**) that is separate from the main computer. Hardware-based RAID devices include separate memory, a CPU, and other components besides the array of hard disks. A hardware-based RAID system contains a separate microprocessor and control circuits; it is practically a separate computer with the sole purpose of providing data to your main system. The main advantage of using a hardware-based RAID system is that all of the special technology is contained in the RAID device. The operating system itself cannot differentiate between a RAID device and any other hard disk. This eliminates the need for any special software or configuration on the Linux system.

Another advantage of hardware RAID systems is that they can provide features that are rarely duplicated without a separate device to house the hard disk array. The most prominent of these features is **hot-swapping**, which makes it possible to pull out and replace a failed hard disk without ever turning the system off. The hot-swapped hard disk is gradually reintegrated into the array as data is written to it; as a result, the failed hard disk entails no downtime. This feature obviously adds to the expense of a RAID system.

A hardware RAID system may include utilities that run on the host operating system and that are designed to manage or configure the RAID device. You must be certain that such utilities can be used on a Linux platform. See the Linux HOWTO document "DPT-Hardware-RAID HOWTO," which explains how to use hardware RAID devices on Linux. The end of the next section describes where to find HOWTO documents.

The main disadvantage of hardware-based RAID is the cost. Because the RAID system has a complete suite of electronics to manage the hard disks, plus a separate case and software utilities, hardware-based RAID is much more expensive than the software-based RAID described next.

Using Software-Based RAID

The term **software-based RAID** refers to RAID that is controlled and managed by software on the computer system that uses the hard disk array. A software-based RAID array includes only the hard disks; the CPU and memory of the computer system to which the array is attached control how the disk array is used. The software used to control this type of RAID array consists of the operating system (the Linux kernel) and additional special device drivers or file system management tools. Because those who use Linux systems are often concerned about the high cost of proprietary computing hardware, software-based RAID is an attractive way for Linux users to achieve the fault tolerance and performance improvements associated with RAID.

To implement software-based RAID on Linux, you can purchase a commercial product designed to support Linux. One such program is the TwinCom duplexed hard disk product (see *www.twincom.com*). Commercial products have the advantage of being fairly easy to install; in addition, the vendors usually provide technical support either for configuration or problem solving.

It is not necessary to choose a commercial version of software-based RAID. The features built into the Linux kernel for support of software-based RAID devices are less expensive and more flexible. However, most system administrators choose not to use these features because they are fairly new and difficult to use. Because these features add fault tolerance without excessive cost (only the extra hard disks and the system administrator's time), you should familiarize yourself with them.

RAID support in the Linux kernel is called Software-RAID. Linux support for softwarebased RAID uses a set of kernel modules called multiple device drivers, which are identified in the /dev directory by the designation md. After enabling the multiple device drivers in the Linux kernel, you can use the RAID software utilities (called `raidtools`) to configure multiple hard disks in accordance with one of the RAID levels described previously. RAID-Linear, RAID-0 (striping), RAID-1 (mirroring/duplexing), and RAID-5 (striping with parity) are supported in the Linux version 2.2 kernels. (This is the kernel shipped with Linux versions

such as Red Hat 6.1, SuSE 6.2, and OpenLinux 2.3.) Figure 9-14 shows the screen from which you can configure RAID options.

A complete discussion of setting up RAID using these kernel features is beyond the scope of this book, but the `raidtools` package and copious additional documentation are available to guide the system administrator in this task. You'll find a great deal of helpful information in the mini-HOWTO called "Software-RAID" (though named "mini," this HOWTO is long and thorough). Regarding the specialized task of using RAID for the root partition of a Linux system, see the HOWTO document called "Root-RAID." These documents are available at many Internet sites where the Linux HOWTO documents are stored, such as *ftp://metalab.unc.edu/pub/Linux/docs/HOWTO/*. Most Linux distributions also include a copy of the HOWTO documents. For example, Red Hat 6.1 includes these files on the first CD-ROM in their package at `/doc/HOWTO` and `/doc/HOWTO/mini`.

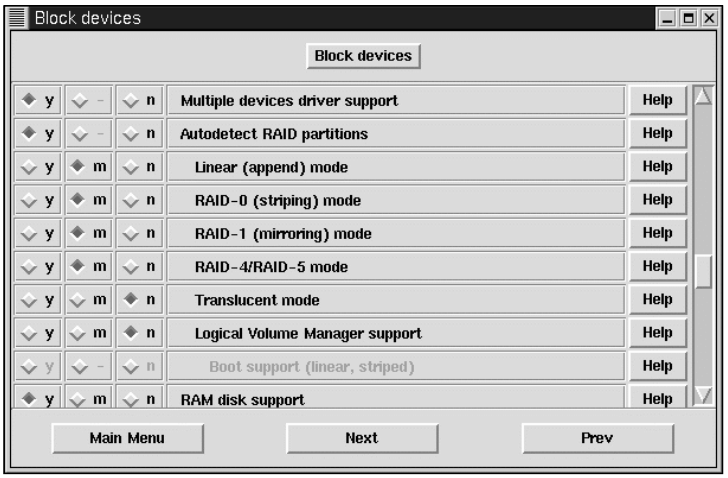

Figure 9-14 Screen used to configure Software-RAID

REDUNDANT SERVERS

The first part of this chapter mentioned using duplicate servers located in different cities as a method of providing a highly fault tolerant information system. However, Linux does not yet have the ability to accommodate redundant servers in the same way that some operating systems can. A redundant server requires very high speed data communications lines and special software to both mirror information and inform each system of when the other is responding to requests and when it should respond because of an outage of some sort. Figure 9-15 shows how two redundant servers might be connected after additional hardware and software support is added to Linux. The connections include:

- A serial cable over which a brief repeating signal (known as a heartbeat) informs each system of the status of the other system. This signal allows special software on each server to know when to take over functions because the other server is

down. (In some server configurations, a high-speed network connection is used instead of a serial cable.)

- A hardware RAID subsystem with a "twin-tailed" cable. The twin-tailed cable allows one RAID subsystem to be connected to two computers. Both computers can access information simultaneously. If either server goes down, the other server still has complete access to the same data.
- Dual network cards in each server, so that if one network card were to fail, the server could still communicate with clients on the network.

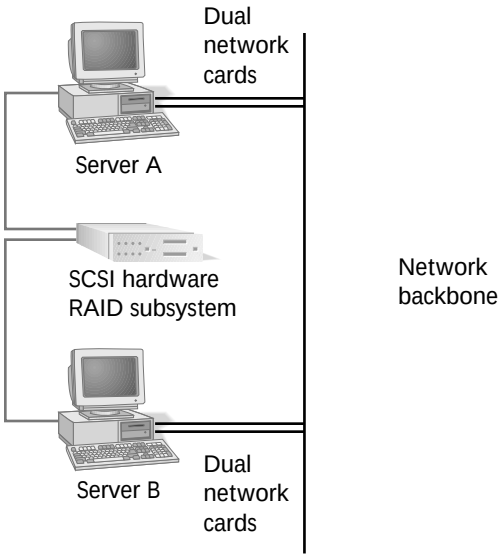

Figure 9-15 Redundant servers

Although standard Linux does not yet support true redundant servers, you can use several technologies to improve the fault tolerance of your Linux systems. Consider the following ideas for improved fault tolerance of Linux systems in large organizations:

- You can use a commercial product such as the TwinCom Linux disk mirroring software mentioned previously to mirror hard disks across a network link. Though this doesn't create a true redundant server that can automatically take over if the other server fails, it does replicate data in two different locations.
- By configuring your router and Domain Name Service appropriately, you can ensure that two or more computers are available to respond to requests for Web pages, FTP information, or other Internet services. Special routers are available to spread incoming requests among several servers that contain identical information. Thus if one server goes down, another server will respond to the incoming requests. This arrangement isn't appropriate for a small system used by only a few people (such as a departmental database server), but it is very helpful (though expensive) for Web sites that cannot tolerate downtime. By using a product like the network-based mirroring in the previous item, you can position these different servers in different offices or cities.

- Some Linux products support a set of multiple servers working in tandem to run Web servers or other network services that require constant uptime. (One notable vendor of such products is Turbo Linux; see the company's Web site at *www.turbolinux.com*.) These additions to a standard Linux distribution manage the transfer of control between systems to handle large numbers of incoming requests. Figure 9-16 illustrates how a DNS name server can route incoming requests for a Web page among a set of Web servers. This setup allows the set of Web servers to handle a large traffic volume with redundancy among the Web servers.

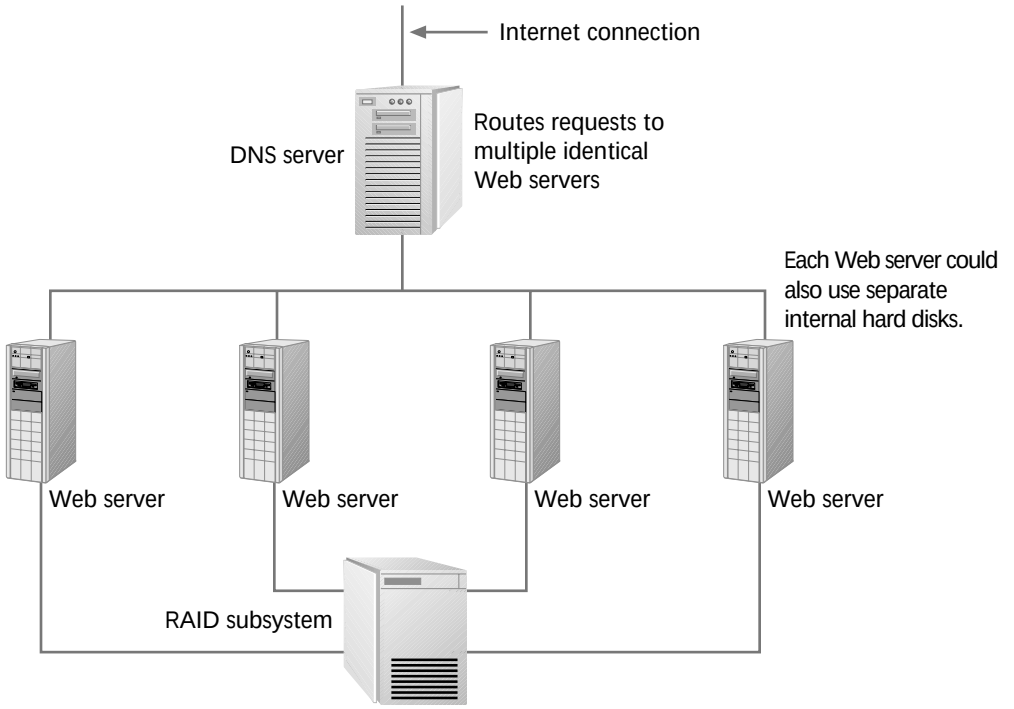

Figure 9-16 A set of Web servers responding to Web page requests

- The high-end server systems sold by the major hardware vendors (such as IBM, Toshiba, Compaq, Dell, and Acer) now support Linux. These servers support features like hot-swap hard disk bays for use with RAID software, redundant power supplies and cooling fans, and other fault tolerant hardware features.

You may have heard of a Linux development called **Beowulf**. This technology allows multiple Linux-based computers to be tied together in a parallel-processing architecture to simulate a supercomputer for work requiring massive amounts of computation (such as calculating weather patterns or world economic models). Beowulf is not intended to create redundancy for a single server, however, because each server in a Beowulf cluster takes on a specific portion of the computing task.

CHAPTER SUMMARY

- Maintaining high availability of a Linux system is increasingly important for many organizations. A disaster plan and an understanding of the hardware and software options available can help reduce downtime for a Linux system. Creating a disk from which a failed system can be booted and maintaining a software master for all server applications are two easy steps that help a system administrator to prepare for possible emergencies.
- The power supply to a computer is a key point of possible failure. By using surge suppressors, maintaining back-up power supply parts, and installing a UPS, the power to a Linux system can be safeguarded.
- Hard disks are the most vulnerable failure point in most computer systems. By checking file systems regularly and using redundant systems such as RAID, information on hard disks can be protected against costly system failures. RAID systems can be controlled by separate hardware devices or directly through software support in the Linux kernel. Several levels of RAID are defined, according to how information is arranged on a set of hard disks. Different RAID levels provide various performance, reliability, and cost trade-offs.

KEY TERMS

AC power — The standard alternating current power coming into a building from a public utility; the power from a wall socket.

array — A collection of multiple hard disks. *See* RAID.

Beowulf — A cluster of multiple Linux servers operating in parallel as a supercomputer to solve complex problems.

block — A unit of storage on a file system. A standard block contains 1024 characters (bytes).

boot disk — A disk used to launch the Linux operating system stored on your hard disk. It can be used in normal operating situations to start the Linux system.

defragmenting — The process of rearranging the files on a file system so that all the parts of a file are located next to each other on the physical hard disk. Defragmenting a hard disk increases system performance and reduces wear on the storage device.

directory record — A file containing the names and inode numbers of other files.

disaster plan — An organized written plan that describes how to respond to various threats to an information system such as Linux.

disk optimization — *See* defragmenting.

downtime — Occasions when an organization's computer systems cannot respond to requests for information.

`dumpe2fs` — A utility used to display technical statistics and parameters about a Linux `ext2` file system.

duplexing — Term used to describe a system in which the contents of two file systems, that are located on different hard disk controllers, contain identical information. Compare to "mirroring," a technique that provides identical information on two file systems but without redundant disk controllers.

`e2defrag` — The utility used to defragment a standard Linux `ext2` partition.

fault tolerance — The condition of being able to tolerate errors or events that might otherwise cause system failure.

file record — An information item within an `ext2` file system that includes a filename and an inode number. The inode itself contains detailed information about the file.

`fsck` — A utility used to check the integrity of an `ext2` file system.

hard link — A pointer to an inode that is already pointed to by at least one other file record.

hardware-based RAID — A RAID array that is contained in a separate hardware device (a RAID subsystem) and is controlled by a CPU and other components separate from the CPU of the Linux system.

high availability — Term used to refer to the processes, products, or programs involved in ensuring that a system experiences as little downtime as possible. The goal for all high availability systems is 100% uptime.

high availability cluster — A group of servers that process the same tasks (resource groups) and take over each others' functionality in the event of an outage or failure.

hot-swapping — The process of removing and replacing a failed hard drive from a RAID hardware device or specialized server without turning off the power to the device.

`init` — A daemon that acts as a control process to start the first processes on Linux, such as the login screens.

inode — A file information record, identified by a unique number within a file system, which contains detailed information about a block of data commonly called a file.

install disk — A disk used when you first install Linux. Its contents are similar to a rescue disk, but it is not designed to assist you in recovering damaged data.

link — A special file record that refers to the same physical file data as another file record.

mirroring — Term used to describe a system in which the contents of two file systems contain identical information. Mirroring improves data access speed and provides fault tolerance in the event that one of the file systems fails.

parity — A technique that allows corrupted data to be reconstructed using an extra piece of information (the parity information) that is created as the data is stored. Parity information provides redundancy for a piece of information.

parity stripe — Parity information stored as part of a RAID-3 or RAID-5 system.

points of failure — Parts of an information system that are subject to failure.

power supply — The component within a computer system that converts the incoming AC power from a wall socket or UPS device to the correct voltage for use by components in a computer.

RAID — A system using multiple inexpensive hard disks arranged in a predefined pattern (an array) to improve performance, increase fault tolerance, or both.

RAID-0 — A RAID level that uses striping to improve disk performance but without adding any fault tolerance.

RAID-1 — A RAID level that uses disk mirroring to significantly improve fault tolerance. Disk read performance is also improved, but disk write performance suffers.

RAID-3 — A RAID level that uses striping with parity information to improve both performance and fault tolerance.

RAID-5 — A RAID level in which data striping with parity is spread across all disks in the RAID array (compared to RAID-3, in which the parity information is stored on a single hard disk).

RAID-Linear — A method of combining multiple physical devices into a single logical device.

RAID subsystem — A hardware-controlled RAID device containing a CPU and other components to control the array of hard disks.

redundancy — Term used to refer to a duplicate system component or piece of data. Many fault tolerant systems rely on the use of redundant components or data; in the event of a failure, the duplicate component or copy of the data would still be available.

redundant arrays of inexpensive disks — *See* RAID.

rescue disk — A disk created specifically to boot a Linux system in the event of a system failure. Contains the software tools most likely to be of help in diagnosing and repairing the problem with the failed system.

rescue mode — A mode of operation in Red Hat Linux that is initiated by starting the system using a rescue disk. Rescue mode is used to repair a system failure that blocks normal booting and operation.

resource groups — The tasks and their accompanying system resources that are defined within a high availability cluster. Each server in the cluster can take over a complete resource group if the server handling that resource group fails.

root disk — A disk used in conjunction with an install disk during some Linux installations (depending on the Linux distribution you are using).

single point of failure — A system component which, if it alone fails, renders a system unusable. Can be a hardware component or a piece of data.

software masters — Original copies of an application supplied by a software vendor or manufacturer, usually one or more CDs, tapes, or disks.

software-based RAID — A RAID array that is controlled or managed by software on the computer system that uses the hard disk array (the Linux system) rather than by a separate CPU or other hardware components.

striping — A technique in which parts of a file are written to more than one disk in order to improve performance. *See* RAID-3.

superblock — The master information record for a Linux file system.

surge suppressor — A device that prevents potentially damaging electrical irregularities from reaching a computer system's power supply.

symbolic link — A file record that includes a path and filename, but not an inode.

`tune2fs` — A utility used to view or adjust parameters within the superblock of a Linux file system.

uninterruptible power supply (UPS) — A device capable of providing power to a computer via batteries when the incoming AC power (wall socket power) fails. It also informs the Linux system of the status of the power.

write caching — A technique in which information to be written to a file system (particularly a RAID file system) is stored in system memory temporarily in order to improve performance of reading and writing information to the file system.

REVIEW QUESTIONS

1. A good disaster plan should consider versus benefit.
   a. RAID level
   b. dangers
   c. cost
   d. availability
2. How does the amount of time that a company can be without data affect the contents of a disaster plan?
3. Why is it important to maintain software masters rather than rely on vendor support?
4. At a minimum, a rescue disk should contain which of the following:
   a. A virtual console option
   b. A text editor and disk repair utilities
   c. A Beowulf configuration
   d. The file system superblock
5. Points of failure in a system have a direct relation to the of the system.
   a. fault tolerance
   b. redundancy
   c. cost
   d. write-caching
6. A built-in back-up power supply is the same thing as an uninterruptible power supply. True or False?
7. A surge suppressor protects against all of the following except:
   a. Power outages
   b. Voltage spikes
   c. Reduced voltage
   d. Brownouts
8. What program is used to monitor a UPS and send a signal indicating that Linux should shut down?

9. A `shutdown` command can be canceled if the main AC power is restored. True or False?
10. The configuration file for the `powerd` daemon includes the following:
    a. Scripts used when shutting down Linux
    b. The serial port and serial pin to monitor
    c. Debug information used to correct system flaws
    d. The call to the `shutdown` command
11. The `init` daemon does which of the following:
    a. Manages all processes and receives signals from the `powerd` daemon
    b. Monitors the UPS via a serial cable
    c. Manages the `powerd.conf` configuration file
    d. Creates a high availability server
12. Which condition is normally *not* monitored by a UPS or the `powerd` daemon?
    a. Main AC power failure
    b. UPS battery low
    c. Main AC power restored
    d. Telephone power failure
13. Redundant arrays of inexpensive disks (RAID) are used to provide:
    a. Lower-cost systems than single disks
    b. Redundant superblock information
    c. Fault tolerance and improved performance
    d. Beowulf clusters
14. Striping refers to which of the following?
    a. Spreading a single file across multiple hard disks
    b. Duplicating file information on multiple hard disks
    c. Adding error-correcting codes to a file
    d. Duplicating inode data
15. Define the advantages of using hot-swapped disk drives.
16. Linux does not yet support any high availability features. True or False?
17. A high availability cluster is a Beowulf cluster used for parallel processing. True or False?
18. Software-based RAID support in Linux is configured as part of which of the following?
    a. Both symbolic and hard links in the file system
    b. The write-caching configuration file
    c. The Linux kernel and related modules and drivers
    d. The `fsck` and `tune2fs` utilities

19. If data contained in a file system's superblock is unreadable, the file system cannot be accessed. True or False?
20. An inode contains information about a file, including all of the following except:
    a. The file's name
    b. The file's location on the hard disk or other device
    c. The file's inode number
    d. Dates and times of creation and last access
21. Symbolic links refer to inode numbers; hard links refer to paths and filenames. True or False?
22. Describe the purpose of reserving a percentage of the space within a file system for the `root` or other user account.
23. At boot time, the `fsck` utility does the following:
    a. Automatically checks each mounted `ext2` file system for errors
    b. Unmounts any damaged file systems
    c. Reads and resets key superblock parameters
    d. Calls the `tune2fs` utility to manage file system start-up
24. The maximum number of times a file system can be mounted between complete `fsck` integrity checks can be configured using `tune2fs`. True or False?
25. Disk fragmentation is not a problem in Linux `ext2` file systems because:
    a. New files are created at locations spread around the entire file system.
    b. Striping used by the Linux Software-RAID feature prevents fragmentation.
    c. The `fsck` utility defragments the file system each time the system is booted.
    d. Fragmentation is not a single point of failure for a high availability system.

HANDS-ON PROJECTS

Project 9-1

In this activity you create a floppy disk containing an `ext2` file system and work with that file system using the `fsck` and `tune2fs` commands. This allows you to experiment with concepts explained in this chapter without damaging the data on a hard disk. To complete this project you need a blank disk and an installed Linux system.

1. Insert a blank 3½-inch disk in the disk drive of your Linux computer.
2. Log in to Linux as `root` and open a command-line window.
3. Format the disk, creating an `ext2` file system, using this command: **`/sbin/mke2fs /dev/fd0`**. Study the information that appears on screen during the formatting process to see what items you recognize from the discussion in the chapter (such as inodes, blocks, and the superbock).

4. Dump the information from the floppy file system using this command: **`/sbin/dumpe2fs /dev/fd0`**. The listing is short because the floppy is a small file system. Review the information found on the superblock to see which fields you recognize from the chapter discussion. (Other fields are described in the `dumpe2fs` manual page.)
5. Change the maximum mount count for the floppy file system using this command: **`/sbin/tune2fs -c 5 /dev/fd0`**. (The default value when you created the file system in Step 3 was 20.)
6. Check that the new value for maximum mount count is stored on the disk by using the `-l` option of the `tune2fs` command to view the superblock (this is similar to the `dumpe2fs` command but only dumps the superblock parameters): **`/sbin/tune2fs -l /dev/fd0`**.
7. Check the integrity of the floppy file system using this command: **`fsck /dev/fd0`**. The results appear within a second or two because the file system is clean.
8. Mount the floppy disk so you can write a file to it, as follows: **`mount -t ext2 /dev/fd0 /mnt/floppy`**. (The `/mnt/floppy` directory exists on most Linux systems. You can create this directory if necessary, or use a different mount point if you wish.)
9. Copy a system file to the floppy disk using this command: **`cp /etc/termcap /mnt/floppy`**. As the file is being written, eject the disk. (A large file is referred to in this example command to give you time to eject the disk while the copy operation is still in progress.) This simulates an error condition or power outage that may damage the file system.
10. Reinsert the floppy disk.
11. Unmount the floppy disk using this command: **`umount /dev/fd0`**.
12. Run the `fsck` command again as follows: **`fsck /dev/fd0`**. This time the command shows the message `/dev/fd0 was not cleanly unmounted, check forced`. Depending on what was in process at the moment you ejected the disk, `fsck` may ask you a question about the disk data, or it may automatically update and correct any damaged status information.
13. Mount the file system again as follows: **`mount -t ext2 /dev/fd0 /mnt/floppy`**.
14. View the file system contents to see if you copied the file successfully, using the following command: **`ls -l /mnt/floppy`**. Because the copy operation was interrupted, the file is probably not there.
15. Unmount the file system again and eject the disk using this command: **`umount /dev/fd0`**.

Project 9-2

In this activity you review information on the `root` file system. This is similar to the previous activity except that you work directly on your hard disk rather than on a blank floppy disk. Follow the instructions carefully to avoid damaging your Linux hard disk. To complete this project you need an installed Linux system.

1. Log in to Linux as `root` and open a command-line window.
2. Enter the **`mount`** command with no options to see which device contains your root file system. The first line of the output from the `mount` command includes this information. The line looks like the following sample output except that the first part (the device name) will be different on your system:

   ```
   /dev/hda3 on / type ext2 (rw)
   ```

3. Use the `dumpe2fs` command to review information about your `root` file system. Pipe the results through the `less` command so you can page through them one screen at a time. The necessary command looks like **`dumpe2fs /dev/hda3 | less`** (substitute the device name for your `root` file system where `/dev/hda3` is used).
4. Look for the line containing `File system volume name` in the output of the `dumpe2fs` command. Is a volume name defined on your system?
5. Look for the line containing `Mount count` in the output of the `dumpe2fs` command. What does this number represent? How does this number relate to the line labeled `Maximum mount count`?
6. Use the `tune2fs` command to list information from the superblock of your `root` file system. Use this command, substituting the device name of your `root` file system: **`tune2fs -l /dev/hda3`**. How does this information compare to the first part of the output from the `dumpe2fs` command?
7. The `tune2fs` command cannot be used to change parameters on a file system that is mounted. Because the `root` file system is always mounted when you boot your system normally, you must boot from a disk (such as a rescue disk) to change parameters on your `root` file system. View the man page of the `tune2fs` command to review what parameters can be changed using this command. Can you identify a hard disk parameter that cannot be changed with `tune2fs`?

Project 9-3

In this activity you review the Software-RAID HOWTO document. This document contains additional technical information about using RAID on Linux. To complete this project you need an Internet connection and a working Web browser.

1. Start your Web browser and go to the Linux HQ site at: *www.linuxhq.com*.
2. Click the link for the Linux Docs/Info section. (You can also explore other sections of this site to learn more about Linux kernel development.)
3. On the Linux Information page, click the link for HOWTOs. Additional documentation links are provided for many Linux-related subjects.
4. On the Linux HOWTO Index page, click **3.2 mini-HOWTOs**. The mini-HOWTOs cover more focused topics; they are not necessarily smaller documents.
5. When the long list of mini-HOWTO documents appears, scroll down and click **Software-RAID mini-HOWTO**. (You may also find many other interesting topics to study on the list of mini-HOWTOs.)

6. Review the sections of the Software-RAID mini-HOWTO document. Click on a section, read or skim some of it, then try the navigation links at the top or bottom of the section to switch to the previous or next section, or back to the table of contents for the Software-RAID mini-HOWTO. What warnings have you noticed about using RAID?
7. In section 1 of the mini-HOWTO, find the list of related Linux RAID documents. Select one that looks interesting and follow it to read more about RAID.

CASE PROJECTS

1. You have been asked to provide consulting services as an expert system administrator to Thompson Mutual Funds, a nationwide financial services company. Thompson employs customer service representatives who respond to customer inquiries about their accounts, via telephone, 24 hours per day. Thompson also maintains a Web site with information about mutual funds and access to stock market pricing information and individual accounts (where customers can view their balance, transfer funds between accounts, and so forth). Thompson has two main offices, one in San Francisco and another in Miami. All of the customer data is stored on Linux-based database servers. Without listing specific hardware and software requirements, what key measures would you use to try to protect Thompson's computer systems and provide high availability? Describe in general terms the features of your disaster plan, what it protects against, what risks it does not fully address, and how costly each part of the plan might be to implement.
2. Considering the strong financial position of Thompson Mutual Funds and the lost value caused by any system downtime, discuss which RAID level or levels you might use in setting up the storage systems on a Linux server within a single Thompson office.
3. The Thompson Web server is located in the San Francisco office. Because existing customers use the Web server to access account information, Thompson wants to avoid any downtime for the Web server. Web server uptime is not as important as keeping the main database servers running, because customers can always call a customer service representative if the Web site is down. Describe the relationship between the Web servers and the database servers. (Is one dependent on the other?) What high availability features would be appropriate for the Web server?